<https://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips>

**Overview**

N-Tier architecture is an industry-proved software architecture model, suitable to support enterprise-level client/server applications by resolving issues like scalability, security, fault tolerance and etc. .NET has many tools and features, but .NET doesn’t have pre-defined ways to guard how to implement N-Tier architecture. Therefore, in order to achieve good design and implementation of N-Tier architecture in .NET, understanding fully its concepts is very important. However, many of us may hear, read or use N-Tier architecture for many years but still misunderstand its concepts more or less. This article tries to clarify many basic concepts in N-Tier architecture from all aspects, and also provide some practical **tips**. The tips in this article are based on the assumption that a team has a full control over all layers of the N-Tier architecture. We have another article to elaborate a N-Tier architecture sample in .NET: [A N-Tier Architecture Sample with ASP.NET MVC3, WCF and Entity Framework](http://www.codeproject.com/Articles/434282/A-N-Tier-Architecture-Sample-with-ASP-NET-MVC3-WCF).

معماری N-Tier یک الگوی معماری نرم افزاری با اثبات صنعت است که برای حل برنامه هایی از قبیل مقیاس پذیری ، امنیت ، تحمل در برابر خطا و غیره ، مناسب برای پشتیبانی از برنامه های مشتری / سرور است. روش های از پیش تعریف شده برای محافظت از نحوه اجرای معماری N-Tier وجود دارد. بنابراین ، برای دستیابی به طراحی خوب و اجرای معماری N-Tier در .NET ، درک کامل مفاهیم آن بسیار مهم است. با این وجود ، بسیاری از ما ممکن است سالها معماری N-Tier را بشنویم ، بخوانیم یا از آن استفاده کنیم اما هنوز مفاهیم آن را کم و بیش درک نمی کنیم. در این مقاله سعی شده بسیاری از مفاهیم اساسی در معماری N-Tier از همه جنبه ها روشن شود و همچنین نکات عملی ارائه شود. نکات موجود در این مقاله بر این فرض استوار است که یک تیم کنترل کاملی بر تمام لایه های معماری N-Tier دارد. مقاله دیگری داریم که می توان نمونه معماری N-Tier را در آن ارائه داد.

### Some Terminology's Difference and Relationship

#### Tier and Layer

Firstly we need to clarify the difference between two terms in N-Tier architecture: tier and layer. Tier usually means the physical deployment computer. Usually an individual running server is one tier. Several servers may also be counted as one tier, such as server failover clustering. By contrast, layer usually means logic software component group mainly by functionality; layer is used for software development purpose. Layer software implementation has many advantages and is a good way to achieve N-Tier architecture. Layer and tier may or may not exactly match each other. Each layer may run in an individual tier. However, multiple layers may also be able to run in one tier.

تفاوت و ارتباط اصطلاحات  
لایه و لایه  
  
در مرحله اول باید تفاوت بین دو اصطلاح را در معماری N-Tier مشخص کنیم: لایه و لایه. Tier معمولاً به معنی کامپیوتر استقرار فیزیکی است. معمولاً یک سرور در حال اجرا فردی یک ردیف است. چندین سرور همچنین ممکن است به عنوان یک ردیف شمرده شوند ، مانند خوشه بندی عدم موفقیت سرور. در مقابل ، لایه معمولاً به معنای گروه کامپوننت نرم افزار منطقی است که عمدتاً از طریق کارایی انجام می شود. لایه برای اهداف توسعه نرم افزار استفاده می شود. اجرای نرم افزار لایه ای مزایای بسیاری دارد و روشی مناسب برای دستیابی به معماری N-Tier است. لایه و لایه ممکن است دقیقاً با یکدیگر مطابقت نداشته باشند. هر لایه ممکن است در یک لایه جداگانه اجرا شود. با این حال ، چندین لایه نیز ممکن است قادر به اجرا در یک لایه باشند.

A layer may also be able to run in multiple tiers. For example, in Diagram 2 below, the persistence layer in .NET can include two parts: persistence Lib and WCF data service, the persistence lib in the persistence layer always runs in the same process as business layer to adapt the business layer to the WCF data service. However, the WCF data service in persistence layer can run in a separate individual tier. Here is another example: we may extract the data validation in business layer into a separate library (but still kept in business layer), which can be called by client presenter layer directly for a better client-side interactive performance. If this occurs, then data validation part of the business layer runs in the same process of the client presenter layer, the rest of business layer runs in a separate tier.

یک لایه همچنین ممکن است قادر به اجرای چندین لایه باشد. به عنوان مثال ، در نمودار 2 در زیر ، لایه ماندگاری در .NET می تواند شامل دو بخش باشد: سرویس داده دایمی Lib و WCF ، lib libstination در لایه persist همیشه در همان فرآیند لایه تجاری اجرا می شود تا لایه تجاری را با WCF تطبیق دهد. خدمات داده با این حال ، سرویس داده WCF در لایه ماندگار می تواند در یک لایه جداگانه اجرا شود. در اینجا مثال دیگری وجود دارد: ما ممکن است اعتبارسنجی داده ها را در لایه کسب و کار در یک کتابخانه جداگانه (اما هنوز در لایه تجاری نگه داشته شده) استخراج کنیم ، که می تواند توسط لایه ارائه دهنده مشتری مستقیماً برای عملکردی بهتر تعاملی از طرف مشتری ایجاد شود. اگر این اتفاق بیفتد ، سپس بخشی از اعتبار سنجی داده ها از لایه کسب و کار در همان فرآیند لایه ارائه دهنده مشتری اجرا می شود ، بقیه لایه کسب و کار در یک لایه جداگانه اجرا می شود.

#### Tier And Process

If a layer can run in an individual process, usually it will also be able to run in an individual computer (tier), hence it can be considered capable for an individual tier in N-Tier architecture. However, this isn’t always true. For example, assume that there are two layers which are implemented to run in two individual processes; they communicate with each other too. However, if these two layers are implemented in a such way that their IPC (inter-process communication) is solely based on a non-distributed way, such as the local shared memory, then these two layers can run in two different processes only in the same computer, not in two different computers. Unless there is another alternative distributed IPC way (such as socket) available for these two layers, these two layers will be considered capable for only one tier even though they can run in two different processes of the same computer.

ردیف و پردازش  
  
اگر یک لایه بتواند در یک فرآیند مجزا اجرا شود ، معمولاً در کامپیوتر شخصی (ردیف) نیز قادر به اجرای آن خواهد بود ، از این رو می توان آن را برای یک ردیف فردی در معماری N-Tier در نظر گرفت. با این حال ، این همیشه درست نیست. به عنوان مثال ، فرض کنید که دو لایه وجود دارد که برای اجرا در دو فرآیند جداگانه اجرا می شوند. آنها با هم ارتباط برقرار می کنند اما اگر این دو لایه به گونه ای پیاده سازی شوند که IPC آنها (ارتباط بین فرآیندی) صرفاً براساس روشی غیر توزیع مانند حافظه مشترک محلی انجام شود ، این دو لایه می توانند فقط در دو فرآیند متفاوت اجرا شوند. همان کامپیوتر ، نه در دو کامپیوتر مختلف. اگر روش IPC جایگزین دیگری (مانند سوکت) برای این دو لایه موجود نباشد ، این دو لایه حتی اگر بتوانند در دو فرآیند مختلف یک کامپیوتر اجرا شوند ، تنها برای یک لایه در نظر گرفته می شوند.

#### Layer and Process

A layer may run in an individual process; several layer may also run in an individual process; a layer may run in several processes too. If you read above section “Tier and Layer Relationship”, you can understand here easily.

لایه و پردازش  
  
یک لایه ممکن است در یک فرآیند فردی اجرا شود. چندین لایه نیز ممکن است در یک فرایند خاص اجرا شوند. یک لایه ممکن است در چندین فرآیند نیز اجرا شود. اگر بخش بالا "روابط لایه و لایه" را می خوانید ، می توانید در اینجا به راحتی درک کنید.

### 3-Tier Architecture

We introduce the 3-Tier concept first so that we can understand other tier concepts later easily. The simplest of N-Tier architecture is 3-Tier which typically contain following software component layers listed from the top level to the low level: presentation layer, application layer and data layer, which are depicted in Diagram 1.

معماری 3-ردیف  
  
ما ابتدا مفهوم 3-ردیف را معرفی می کنیم تا بتوانیم بعداً سایر مفاهیم سطح را به راحتی درک کنیم. ساده ترین معماری N-Tier 3-Tier است که به طور معمول شامل لایه های مؤلفه نرم افزاری زیر است که از سطح بالا تا سطح پایین ذکر شده است: لایه ارائه ، لایه کاربرد و لایه داده ، که در نمودار 1 به تصویر کشیده شده است.

A layer can access directly only the public components of its directly-below layer. For example, presentation layer can only access the public components in application layer, but not in data layer. Application layer can only access the public components in data layer, but not in presentation layer. Doing so can minimize the dependencies of one layer on other layers. This dependency minimization will bring benefits for layer development/maintenance, upgrading, scaling and etc. Doing so also makes the tier security enforcement possible. For example, the client layer cannot access the data layer directly but through the application layer, so data layer has a higher security guarding. Finally, doing so can also avoid cyclic dependencies among software components.

یک لایه می تواند مستقیماً به مؤلفه های عمومی لایه مستقیم زیر آن دسترسی داشته باشد. به عنوان مثال ، لایه ارائه فقط می تواند به لایه های برنامه به مؤلفه های عمومی دسترسی پیدا کند ، اما نه در لایه داده. لایه برنامه فقط می تواند به لایه های داده به مؤلفه های عمومی دسترسی پیدا کند ، اما نه در لایه ارائه. انجام این کار می تواند وابستگی یک لایه به لایه های دیگر را به حداقل برساند. این به حداقل رساندن وابستگی مزایایی برای توسعه / نگهداری لایه ها ، ارتقاء ، مقیاس گذاری و غیره خواهد داشت. انجام این کار باعث می شود اجرای امنیت لایه نیز امکان پذیر شود. به عنوان مثال ، لایه مشتری نمی تواند مستقیماً به لایه داده دسترسی داشته باشد اما از طریق لایه برنامه ، بنابراین لایه داده از امنیت بیشتری برخوردار است. سرانجام ، انجام این کار همچنین می تواند از وابستگی حلقوی در بین اجزای نرم افزار جلوگیری کند.

In order to claim a complete 3-Tier architecture, all three layers should be able to run in separate computers.

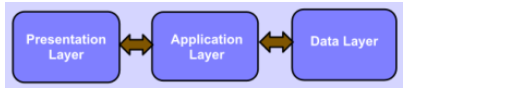
برای ادعای معماری کامل 3 لایه ، هر سه لایه باید بتوانند در رایانه های جداگانه اجرا شوند.

Diagram 1: 3-Tier Architecture

These three layers are briefly described as below:

نمودار 1: معماری 3 لایه  
این سه لایه به طور خلاصه به شرح زیر شرح داده شده است:

**Presentation layer**: a layer that users can access directly, such as desktop UI, web page and etc. Also called client.

لایه ارائه: لایه ای است که کاربران می توانند به طور مستقیم به آن دسترسی پیدا کنند ، از جمله رابط کاربری رومیزی ، صفحه وب و غیره که همچنین به آن مشتری گفته می شود.

**Application layer**: this layer encapsulates the business logic (such as business rules and data validation), domain concept, data access logic and etc. Also called middle layer.

لایه کاربرد: این لایه منطق تجارت (مانند قوانین کسب و کار و اعتبار سنجی داده ها) ، مفهوم دامنه ، منطق دسترسی به داده ها و غیره را محصور می کند که به آن لایه میانی نیز می گویند.

**Data layer**: the external data source to store the application data, such as database server, CRM system, ERP system, mainframe or other legacy systems and etc. The one we meet often today is database server. For N-Tier architecture, we need to use the non-embedded database server, such as SQL server, Oracle, DB2, MySQL or PostgreSQL. The non-embedded database server can be run in an individual computer. Whereas, the embedded type databases, such as Microsoft access, dbase and etc, cannot run in an individual computer, and then cannot be used as the data layer of the 3-Tier architecture.

لایه داده: منبع داده خارجی برای ذخیره داده های برنامه مانند سرور بانک اطلاعاتی ، سیستم CRM ، سیستم ERP ، سیستم اصلی یا سایر سیستم های میراث و غیره است. برای معماری N-Tier ، ما باید از سرور پایگاه داده غیر جاسازی شده ، مانند سرور SQL ، Oracle ، DB2 ، MySQL یا PostgreSQL استفاده کنیم. سرور پایگاه داده غیر جاسازی شده می تواند در یک کامپیوتر شخصی اجرا شود. در حالی که ، بانکهای اطلاعاتی از نوع تعبیه شده ، مانند دسترسی مایکروسافت ، dbase و غیره ، نمی توانند در یک رایانه شخصی اجرا شوند ، و سپس نمی توانند به عنوان لایه داده معماری 3 لایه مورد استفاده قرار گیرند.

### 1, 2, 3 or More Tier Architecture

**1-Tier**: all above layers can only run in one computer. In order to achieve 1-Tier, we need to use the embedded database system, which cannot run in an individual process. Otherwise, there will be at least 2-Tier because non-embedded databases usually can run in an individual computer (tier).

**2-Tier**: either presentation layer and application layer can only run in one computer, or application layer and data layer can only run in one computer. The whole application cannot run in more than 2 computers.

**3-Tier**: the simplest case of N-Tier architecture; all above three layers are able to run in three separate computers. Practically, these three layers can also be deployed in one computer (3-Tier architecture, but deployed as 1-Tier).

**N-Tier**: 3 or more tiers architecture. Diagram 2 below depicts a typical N-Tier architecture. Some layers in 3-Tier can be broken further into more layers. These broken layers may be able to run in more tiers. For example, application layer can be broken into business layer, persistence layer or more. Presentation layer can be broken into client layer and client presenter layer. In diagram 2, in order to claim a complete N-Tier architecture, client presenter layer, business layer and data layer should be able to run in three separate computers (tiers). Practically, all these layers can also be deployed in one compute (tier).

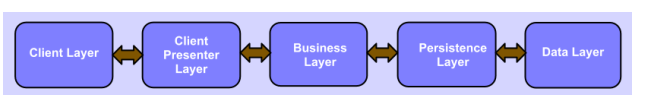
N-Tier: معماری 3 یا چند لایه. نمودار 2 معماری معمولی N-Tier را نشان می دهد. بعضی از لایه ها در 3 لایه می توانند به لایه های بیشتری شکسته شوند. این لایه های شکسته ممکن است در لایه های بیشتری اجرا شوند. به عنوان مثال ، لایه کاربرد را می توان به لایه تجاری ، لایه ماندگاری یا موارد دیگر تقسیم کرد. لایه ارائه می تواند به لایه مشتری و لایه ارائه دهنده مشتری تقسیم شود. در نمودار 2 ، برای ادعای معماری کامل N-Tier ، لایه ارائه دهنده مشتری ، لایه کسب و کار و لایه داده باید بتواند در سه کامپیوتر جداگانه (ردیف) اجرا شود. به طور عملی ، همه این لایه ها نیز می توانند در یک محاسبه (لایه) مستقر شوند.

Diagram 2: N-Tier Architecture

Below are brief summaries on all layers in Diagram 2: در زیر خلاصه ای از همه لایه های نمودار 2 آورده شده است:

Client layer: this layer is involved with users directly. There may be several different types of clients coexisting, such as WPF, Window form, HTML web page and etc.

Client layer: این لایه بطور مستقیم با کاربران درگیر می شود. ممکن است چندین نوع مختلف مشتری وجود داشته باشند ، مانند WPF ، فرم پنجره ، صفحه وب HTML و غیره.

Client presenter layer: contains the presentation logic needed by clients, such as ASP .NET MVC in IIS web server. Also it adapts different clients to the business layer.

لایه ارائه دهنده مشتری: حاوی منطق ارائه مورد نیاز مشتری مانند ASP .NET MVC در وب سرور IIS است. همچنین مشتری های مختلفی را با لایه تجاری سازگار می کند.

Business layer: handles and encapsulates all of business domains and logics; also called domain layer.

لایه کسب و کار: کلیه حوزه ها و منطق تجاری را کنترل و محاصره می کند. لایه دامنه نیز نامیده می شود.

Persistence layer: handles the read/write of the business data to the data layer, also called data access layer (DAL).

لایه ماندگاری: خواندن / نوشتن داده های کسب و کار را به لایه داده ، که به آن لایه دسترسی داده ها نیز گفته می شود (DAL) می دهد.

Data layer: the external data source, such as a database.

لایه داده: منبع داده خارجی مانند پایگاه داده است.

Sometimes, the number of tiers is able to be equal or more than 3, but client presenter layer, business layer and data layer cannot run in three separate computers (tiers). Is this a N-Tier architecture? we categorize this N-Tier as an incomplete N-Tier architecture because its client presenter layer, business layer and data layer cannot run in three separate computers (tiers).

بعضی اوقات ، تعداد ردیف ها می توانند برابر یا بیشتر از 3 باشند ، اما لایه ارائه دهنده مشتری ، لایه کسب و کار و لایه داده در سه رایانه جداگانه (لایه) قابل اجرا نیست. آیا این یک معماری N-Tier است؟ ما این N-Tier را به عنوان یک معماری ناقص N-Tier طبقه بندی می کنیم زیرا لایه ارائه دهنده مشتری ، لایه کسب و کار و لایه داده آن نمی تواند در سه کامپیوتر جداگانه (ردیف) اجرا شود.

If we use the modem non-embedded database such as Sql Server, Oracle and etc, these databases will always be able to run in an individual computer. Therefore, for this case in Diagram 1, the criteria of a 2-Tier architecture is that presentation layer and application layer can run in only one computer; the criteria of a complete 3-Tier architecture is that presentation layer and application layer can run in different computers. A complete N-Tier architecture has the same criteria as 3-Tier.

گر از دیتابیس غیر جاسازی شده مودم مانند Sql Server ، Oracle و غیره استفاده کنیم ، این پایگاه داده ها همیشه قادر به اجرای یک رایانه شخصی هستند. بنابراین ، برای این مورد در نمودار 1 ، معیارهای معماری 2-ردیف این است که لایه ارائه و لایه برنامه فقط در یک کامپیوتر قابل اجرا هستند. معیارهای معماری کامل 3 لایه این است که لایه ارائه و لایه برنامه می توانند در رایانه های مختلف اجرا شوند. یک معماری کامل N-Tier همان معیارهای 3-Tier را دارد.

## Advantages and Disadvantages of Different Tier Architectures

### **1 or 2-Tier Architecture**

**Advantages**: simple and fast for a lower number of users due to fewer processes and fewer tiers; low cost for hardware, network, maintenance and deployment due to less hardware and network bandwidth needed.

**Disadvantages**: will have issues when the number of users gets big; has limitation to solve issues like security, scalability, fault tolerance and etc because it can be deployed in only 1 or 2 computes.

## معماری 1 یا دو لایه

## مزایا: ساده و سریع برای تعداد کمتری از کاربران به دلیل پردازش کمتر و تعداد ردیف کمتر. به دلیل کمبود سخت افزار و پهنای باند شبکه ، هزینه سخت افزاری ، شبکه ، نگهداری و استقرار کم هزینه است.

## معایب: وقتی تعداد کاربران زیاد شود ، مشکلاتی به وجود می آید. محدودیتی برای حل مسائلی مانند امنیت ، مقیاس پذیری ، تحمل گسل و غیره دارد زیرا فقط در یک یا دو محاسبه قابل استفاده است.

### **N-Tier Architecture**

## **Advantages**: there are following general advantages:

1-Scalable: this is due to its capability of multiple tier deployment and the tier decoupling it brought. For example, the data tier can be scaled up by database clustering without other tiers involving. The web client side can be scaled up by load-balancer easily without affecting other tiers. Windows server can be clustered easily for load balancing and failover. In addition, business tier server can also be clustered to scale up the application, such as Weblogic cluster in J2EE.

مقیاس پذیر: این به دلیل قابلیت استقرار چند لایه و جابجایی سطح آن است. به عنوان مثال ، طبقه بندی داده ها را می توان با طبقه بندی پایگاه داده بدون طبقه بندی های دیگر درگیر کرد. طرف مشتری تحت وب می تواند به راحتی و بدون تأثیرگذاری بر سایر ردیف ها توسط بالانساز بار کوچک شود. سرور ویندوز را می توان به راحتی برای توازن بار و عدم موفقیت دسته بندی کرد. علاوه بر این ، سرور ردیف تجاری نیز می تواند برای مقیاس کردن برنامه ، مانند خوشه Weblogic در J2EE ، خوشه بندی شود.

2-Better and finer security control to the whole system: we can enforce the security differently for each tier if the security requirement is different for each tier. For example, business tier and data tier usually need higher security level than presentation tier does, then we can put these two high security tiers behind firewall for protection. 1 or 2 tiers architecture cannot fully achieve this purpose because of a limited number of tiers. Also, for N-Tier architecture, users cannot access business layer and data layer directly, all requests from users are routed by client presenter layer to business layer, then to data layer. Therefore, client presenter layer also serves as a proxy-like layer for business layer, and business layer serves as a proxy-like layer for data layer. These proxy-like layers provides further protection for their layers below.

کنترل امنیتی بهتر و دقیق تر برای کل سیستم: اگر شرایط امنیتی برای هر لایه متفاوت باشد ، می توانیم امنیت را برای هر لایه متفاوت اعمال کنیم. به عنوان مثال ، ردیف تجاری و سطح داده معمولاً به سطح امنیتی بالاتری نسبت به سطح ارائه نیاز دارند ، بنابراین می توانیم این دو لایه امنیتی بالا را در پشت دیوار آتش برای محافظت قرار دهیم. معماری 1 یا 2 لایه به دلیل تعداد محدودی از رده ها به طور کامل نمی تواند به این هدف برسد. همچنین ، برای معماری N-Tier ، کاربران نمی توانند مستقیماً به لایه کسب و کار و لایه داده دسترسی پیدا کنند ، کلیه درخواست های کاربران توسط لایه ارائه دهنده مشتری به لایه تجاری و سپس به لایه داده هدایت می شود. بنابراین ، لایه ارائه دهنده مشتری همچنین به عنوان یک لایه پروکسی مانند برای لایه کسب و کار عمل می کند ، و لایه کسب و کار به عنوان یک لایه پروکسی مانند برای لایه داده عمل می کند. این لایه های پروکسی مانند محافظت بیشتری را برای لایه های زیر آنها ایجاد می کنند.

3-Better fault tolerance ability: for example, the databases in data layer can be clustered for failover or load balance purpose without affecting other layers.

توانایی تحمل بهتر خطا: به عنوان مثال ، پایگاه داده های موجود در لایه داده ها می توانند برای عدم موفقیت و یا عدم تعادل بار بدون تأثیرگذاری بر لایه های دیگر ، دسته بندی شوند.

4-Independent tier upgrading and changing without affecting other tiers: in object-oriented world, Interface-dependency implementation can decouples all layers very well so that each layer can change individually without affecting other layers too much. Interface-dependency means a layer depends on another layer by interfaces only, not concrete classes. Also, the dependency of a layer only on its directly-below layer also minimizes the side effect of a layer’s change on the whole system. For example, if keep the interfaces unchanged, we can update or replace the implementation of any layer independently without affecting the whole system. Due to the changing of business requirement and technology, changing the implementation of a layer to another totally different one does happen often. For example, originally we use Windows Form mainly, now we use WPF mainly. If our original system is implemented as the decoupled layer structure, then we will only need to update the client side from Windows Form to WPF without the need to change the server side layers.

ارتقاء و تغییر لایه مستقل بدون تأثیرگذاری بر سایر رده ها: در دنیای شی گرا ، اجرای وابستگی به رابط می تواند همه لایه ها را به خوبی جدا کند ، به طوری که هر لایه می تواند بطور جداگانه تغییر کند بدون اینکه بیش از حد روی لایه های دیگر تأثیر بگذارد. وابستگی به رابط یعنی یک لایه فقط به لایه های دیگر فقط با رابط ها بستگی دارد ، نه کلاس های بتونی. همچنین ، وابستگی یک لایه فقط به لایه مستقیم زیر آن نیز اثر جانبی تغییر یک لایه در کل سیستم را به حداقل می رساند. به عنوان مثال ، اگر واسط ها را بدون تغییر نگه داریم ، می توانیم اجرای هر لایه را بطور مستقل به روزرسانی یا جایگزین کنیم بدون اینکه روی کل سیستم تأثیر بگذارد. با توجه به تغییر نیاز و فن آوری کسب و کار ، تغییر اجرای یک لایه به حالت کاملا متفاوت اغلب اتفاق می افتد. به عنوان مثال ، در ابتدا ما از Windows Form به طور عمده استفاده می کنیم ، اکنون از WPF استفاده می کنیم. اگر سیستم اصلی ما به عنوان ساختار لایه جدا شده پیاده سازی شود ، ما فقط نیاز به بروزرسانی سمت مشتری از Windows Form به WPF بدون نیاز به تغییر لایه های سمت سرور خواهیم داشت.

5-Friendly and efficient for development: the decoupled layers are logic software component groups mainly by functionality, they are very software development friendly and efficient. Each layer can be assigned individually to a team who specializes in the specific functional area; a specialized team can handle the relevant task better and more efficiently.

دوستانه و کارآمد برای توسعه: لایه های جداشده گروه های نرم افزاری منطقی هستند که عمدتاً از لحاظ عملکردی ، از نظر توسعه نرم افزار بسیار دوستانه و کارآمد هستند. هر لایه را می توان به طور جداگانه به تیمی اختصاص داد که در حوزه عملکردی خاصی تخصص دارد. یک تیم تخصصی می تواند وظیفه مربوط را بهتر و مؤثرتر انجام دهد.

6-Friendly for maintenance: N-Tier architecture groups different things together mainly by functionality and then makes things clear, easily understandable and manageable.

دوستانه برای نگهداری: معماری N-Tier موارد مختلفی را با همدیگر عمدتاً از طریق قابلیت های مختلف گروه بندی می کند و سپس امور را روشن ، به راحتی قابل درک و کنترل می کند.

7-Friendly for new feature addition: due to the logical grouped components and the decoupling brought by N-Tier architecture, new features can be added easily without affecting too much on the whole system.

دوستانه برای علاوه بر ویژگی های جدید: با توجه به مؤلفه های گروه بندی منطقی و جداشدگی ناشی از معماری N-Tier ، ویژگی های جدید را می توان به راحتی و بدون تأثیر بیش از حد بر کل سیستم اضافه کرد.

8-Better reusability: this is due to the logically grouped components and the loose couplings among layers. Loosely-coupled component groups are usually implemented in more general ways, so they can be reused by more other applications.

قابلیت استفاده مجدد بهتر: این به دلیل مؤلفه های منطقی طبقه بندی شده و اتصالات شل بین لایه ها است. گروه های کاملاً مؤثر به هم پیوسته معمولاً به روشهای عمومی تری اجرا می شوند ، بنابراین در برنامه های کاربردی دیگر می توان از آنها استفاده مجدد کرد.

#### **The Disadvantages of the N-Tier Deployment**

1. The performance of the whole application may be slow if the hardware and network bandwidth aren’t good enough because more networks, computers and processes are involved.
2. More cost for hardware, network, maintenance and deployment because more hardware and better network bandwidth are needed.

1- اگر سخت افزار و پهنای باند شبکه به اندازه کافی خوب نباشند ، عملکرد کل برنامه ممکن است کند باشد زیرا شبکه ها ، رایانه ها و پردازش های بیشتری درگیر هستند.

2- هزینه سخت افزاری ، شبکه ، نگهداری و استقرار هزینه بیشتری به دلیل سخت افزار بیشتر و پهنای باند بهتر شبکه مورد نیاز است.

The effects of N-Tier deployment on the application performance are a double edge issue. In one side, if the number of uses isn’t big enough, the performance may be slow due to more computers, process and network involved. Namely, if put everything in one tier or one process, performance will be better for a small number of users. However, if the number of user gets big, then the scalability brought by N-Tier will improve the overall performance, such as load balancing and database clustering which all improve the performance of N-Tier architecture. Why the performance result of the small number of users and the big number of users are different? this is because the bottleneck of the whole application are different for these two situations. For the case with the small number users, the bottleneck is the time for data to communicate among different processes. If more computer, more processes and longer network, then costs longer, then performance is bad. However, when the number of user gets big, the bottleneck shift to other things because of server’s capacity, such as cpu and memory resource contentions in one computer, database threshold in server, horsepower limitation of a web server and etc. Only the scalability of N-Tier architecture can solve these bottlenecks existing with a big number of users; usually load balancing by server clustering is used for N-Tier architecture’s scalability. With more computers are scaled up to share the task for big number of users, then performance is improved.  Besides gaining performance by scalability of N-Tier architecture, we can also improve the performance with better hardware and better network bandwidth to meet our business requirement.

تأثیرات استقرار N-Tier بر عملکرد برنامه یک مسئله دو لبه است. در یک طرف ، اگر تعداد استفاده ها به اندازه کافی بزرگ نباشد ، به دلیل درگیری بیشتر رایانه ها ، پردازش ها و شبکه ، عملکرد ممکن است کند باشد. یعنی ، اگر همه چیز را در یک مرحله یا یک فرآیند قرار دهید ، عملکرد برای تعداد کمی از کاربران بهتر خواهد بود. اما اگر تعداد کاربر زیاد شود ، مقیاس پذیری حاصل از N-Tier باعث بهبود عملکرد کلی می شود ، مانند توازن بار و خوشه بندی پایگاه داده که همگی باعث بهبود عملکرد معماری N-Tier می شوند. چرا نتیجه عملکرد تعداد کمی از کاربران و تعداد زیاد کاربران متفاوت است؟ دلیل این است که تنگنا کل برنامه برای این دو وضعیت متفاوت است. در مورد کاربران کم تعداد ، تنگناها زمان ارتباط داده ها در بین فرآیندهای مختلف است. اگر بیشتر رایانه ، پردازش های بیشتر و شبکه طولانی تر باشد ، پس هزینه آن طولانی تر است ، عملکرد آن بد است. اما وقتی تعداد کاربر زیاد شود ، تنگناها به دلیل ظرفیت سرور از قبیل cpu و منابع منبع حافظه در یک رایانه ، آستانه بانک اطلاعاتی در سرور ، محدودیت اسب بخار یک سرور وب و غیره به موارد دیگر تغییر می یابد و فقط مقیاس پذیری معماری N-Tier می تواند این تنگناهای موجود را با تعداد زیادی از کاربران حل کند. معمولاً برای مقیاس پذیری معماری N-Tier از توازن بار توسط خوشه بندی سرور استفاده می شود. با رایانه های بیشتری برای به اشتراک گذاشتن کار در تعداد زیادی از کاربران ، برنامه ریزی شده و عملکرد آن بهبود می یابد. علاوه بر به دست آوردن عملکرد با مقیاس پذیری معماری N-Tier ، ما همچنین می توان عملکرد را با سخت افزار بهتر و پهنای باند شبکه بهتر برای برآورده کردن نیازهای تجاری خود بهبود بخشید.

## Business Data Validation in N-Tier Architecture

Data validation is important and a MUST in N-Tier architecture in order to keep the whole business system healthy and integral. The first question for business data validation will be: where or which layer should handle the data validation? There are some rules and facts as below for the business data validation, which will give us some tips and can answer this question too:

اعتبارسنجی داده ها مهم است و باید در معماری N-Tier به منظور حفظ کل سیستم تجاری سالم و یکپارچه باشد. اولین سوال برای اعتبارسنجی داده های تجاری این خواهد بود: کجا یا کدام لایه باید اعتبارسنجی داده ها را اداره کند؟ برخی از قوانین و حقایق به شرح زیر برای اعتبارسنجی داده های تجاری وجود دارد که نکاتی را به ما ارائه می دهد و می تواند به این سؤال نیز پاسخ دهد:

1. Data validation can be checked in any layer. Usually, the closer to the client layer the validation is, the more efficient the performance is. The farther to the client layer the validation is, the more reliable and robust the application is. When the validation is checked in business layer or persistence layer, it is guaranteed that every type client will get the validation regardless whether or not the client side will check the validation.

1-اعتبارسنجی داده ها در هر لایه قابل بررسی است. معمولاً ، هرچه اعتبار سنجی به لایه مشتری نزدیکتر باشد ، عملکرد آن نیز کارآمدتر است. اعتبار سنجی هر چه دورتر به لایه مشتری باشد ، قابلیت اطمینان و استحکام بیشتری دارد. وقتی اعتبار سنجی در لایه تجاری یا لایه ماندگاری بررسی شود ، تضمین می شود که هر مشتری نوع اعتبارسنجی را بدون توجه به اینکه طرف مشتری اعتبارسنجی را بررسی کند یا خیر ، دریافت می کند.

1. When we decide which layer should do the validation, we need to achieve a balance result between performance, reliability and robustness, also we need to make decisions based on the actual situation. If we have full control of the all layers, we can let all validations happen in client/client presenter layers only to gain performance. However, if the business layer is also exposed to some client/client presenter layers which is out of our control, then the business or lower layer must do all validations to gain reliability regardless of whether or not our clients do the same validation.

2- هنگامی که تصمیم می گیریم کدام لایه باید اعتبارسنجی را انجام دهد ، باید به نتیجه توازن بین عملکرد ، قابلیت اطمینان و استحکام برسیم ، همچنین باید بر اساس وضعیت واقعی تصمیم گیری کنیم. اگر کنترل کامل همه لایه ها را داشته باشیم ، می توانیم اجازه دهیم که تمام اعتبارسنجی ها در لایه های ارائه دهنده مشتری / مشتری فقط برای به دست آوردن عملکرد اتفاق بیفتد. اما اگر لایه کسب و کار نیز در معرض برخی از لایه های ارائه دهنده مشتری / مشتری قرار دارد که خارج از کنترل ما است ، پس کسب و کار یا لایه پایین باید تمام اعتبارسنجی ها را انجام دهد تا اعتماد به نفس را بدون توجه به اینکه مشتریان ما همان اعتبار را انجام می دهند یا خیر ، انجام دهند.

1. Client side validation is efficient, such as Javascript validation in a web page. However, users may bypass the client side validation easily and intentionally, such as webpage hacking. Therefore, it is needed to do data validation in both client side and server side to achieve both performance and reliability. The business layer and other further lower layers usually belong to the server side. Client presenter layer may or may not be in the server side; a webserver client presenter layer such as ASP.NET is in the server side. Client presenter layer for WPF may not be in the server side.

3. اعتبارسنجی سمت مشتری کارآمد است ، مانند اعتبار Javascript در یک صفحه وب. با این حال ، کاربران ممکن است اعتبار طرف مشتری را به راحتی و عمدی از قبیل هک صفحه وب دور بزنند. بنابراین برای دستیابی به عملکرد و قابلیت اطمینان لازم است که اعتبارسنجی داده ها را هم در سمت مشتری و هم در سمت سرور انجام دهید. لایه کسب و کار و سایر لایه های پایین تر دیگر معمولاً متعلق به سمت سرور است. لایه ارائه دهنده مشتری ممکن است در سمت سرور باشد یا نباشد. یک لایه ارائه دهنده سرویس دهنده وب سرور مانند ASP.NET در سمت سرور است. لایه ارائه دهنده مشتری برای WPF ممکن است در سمت سرور نباشد.

1. A more practical way is to do the simple data validation in client side for performance and further do a full validation in server side for reliability. The simple data validation is mainly the single property checking of an entity instance. The full validation includes the simple data validation and some complex data validation. The complex data validation can include class-level data validation which crosses multiple properties of an entity instance and the data validation which crosses multiple entity instances of similar or different types.

4- یک روش عملی تر انجام اعتبار سنجی داده های ساده در سمت مشتری برای عملکرد و انجام اعتبارسنجی کامل در سمت سرور برای قابلیت اطمینان است. اعتبارسنجی داده های ساده عمدتاً بررسی خاصیت دارایی یک نمونه از موجودیت است. اعتبارسنجی کامل شامل اعتبارسنجی داده های ساده و اعتبار سنجی داده های پیچیده است. اعتبارسنجی داده های پیچیده می تواند شامل اعتبارسنجی داده های سطح کلاس باشد که از خواص متعدد یک مثال موجودیت و اعتبارسنجی داده ها عبور می کند که از چندین نمونه از انواع مختلف یا مشابه عبور می کند.

1. For some interactive client application, we need to do client side validation anyway for acceptable interactive performance, regardless of whether or not we will do the validation in server side.  Some game applications belong to this category.

5- برای برخی از برنامه های مشتری تعاملی ، لازم است اعتبار سنجی سمت مشتری را به هر حال برای عملکرد قابل قبول تعاملی انجام دهیم ، صرف نظر از اینکه اعتبار را در سمت سرور انجام خواهیم داد یا خیر. برخی از برنامه های بازی به این دسته تعلق دارند.

1. We should implement and maintain one version of validation logics in one place, regardless of where the validation will be checked. All layers should share this one version of validation logics. Why? Doing so has better reusability; it can avoid duplicate and conflicting validation logic in many places and makes the development, maintenance and deployment easier; it keeps the whole validation logics consistent throughout the whole application. In addition, the places to check validation may change with the changing and growing of the business, so validation should be kept in one place with one version but flexible to be called by any layer if needed.

6- ما باید یک نسخه از منطق های اعتبار سنجی را در یک مکان پیاده سازی و نگهداری کنیم ، صرف نظر از اینکه در آن صحت سنجی بررسی می شود. همه لایه ها باید این نسخه منطق اعتبار را به اشتراک بگذارند. چرا؟ انجام این کار قابلیت استفاده مجدد بهتری دارد. این امر می تواند در بسیاری از مناطق از منطق اعتبار سنجی کپی و متناقض جلوگیری کند و توسعه ، نگهداری و استقرار را آسان تر کند. کل منطق اعتبار سنجی را در طول کل برنامه حفظ می کند. علاوه بر این ، مکانهایی برای بررسی اعتبار سنجی ممکن است با تغییر و رشد کار تغییر کند ، بنابراین اعتبار سنجی باید در یک مکان با یک نسخه نگهداری شود اما انعطاف پذیر باشد تا در صورت لزوم توسط هر لایه فراخوانی شود.

## How to Deploy N-Tier Application Correctly

More tiers brings extra complexity, extra deployment/maintenance effort and extra cost. Therefore, the number of tiers should be kept as minimal as enough to solve issues like the scalability, security, failover and etc. If these issues are solved as needed, don’t deploy more tiers further. But, in order to solve these issues as better as possible, usually 3-Tier will be needed at least. If these issues aren’t concerned at all in certain cases, then we can select 1 or 2-Tier architecture or 1 or 2-Tier deployment of N-Tier architecture to gain performance. What is the best number of tiers? no fixed answer. In order to meet our business requirement, we need to select the number of tiers to achieve a best balance result among those good and bad things of N-Tier architecture.

ردیف های بیشتر پیچیدگی اضافی ، تلاش بیشتری برای استقرار و نگهداری و هزینه های اضافی را به همراه دارد. بنابراین ، تعداد ردیف ها باید به اندازه کافی حفظ شود تا مسائلی مانند مقیاس پذیری ، امنیت ، عدم موفقیت و غیره حل شود. اگر این مسائل در صورت نیاز حل شد ، تعداد بیشتری ردیف ها را مستقر نکنید. اما ، برای حل هرچه بهتر این مسائل ، معمولاً حداقل به 3 لایه مورد نیاز است. اگر این موارد در موارد خاص به هیچ وجه مورد توجه قرار نمی گیرند ، می توانیم از معماری 1 یا 2-ردیف یا استقرار 1 یا 2-طبقه از معماری N-Tier برای عملکرد استفاده کنیم. بهترین تعداد لایه ها چیست؟ بدون جواب ثابت. برای رسیدن به نیاز تجاری خود ، باید تعداد ردیف ها را انتخاب کنیم تا به بهترین نتیجه تعادل بین آن چیزهای خوب و بد معماری N-Tier برسیم.

We should differ the following two cases: a) all layers run in one process of a computer, and b) all layers run in different processes of a computer. The 1st case is actually 1 tier architecture; the 2nd case is usually N-Tier architecture but deployed in just 1 tier. Even both are in one computer, the 1 tier architecture will have better performance because of fewer processes involved. Compared to one process, communication crossing process boundaries is more complex and slower, regardless what type of IPC (inter-process communication) technique is used: TCP/IP, named pipe, message queue or shared memory and etc. Therefore, in a deployed computer, we need to keep the number of the application processes as few as possible to gain performance. How to achieve this? the N-Tier architecture can be implemented in such a way that switching among different tier architectures is as easy as updating the configuration file only. This is introduced in detail in our sample application article: A N-Tier Architecture Sample with ASP.NET MVC3, WCF and Entity Framework.

ما باید دو مورد زیر را با هم فرق کنیم: الف) همه لایه ها در یک فرآیند یک کامپیوتر اجرا می شوند و ب) همه لایه ها در فرآیندهای مختلف یک کامپیوتر اجرا می شوند. مورد اول در واقع 1 معماری ردیف است. مورد دوم معمولاً معماری N-Tier است اما فقط در یک ردیف مستقر می شود. حتی هر دو در یک رایانه قرار دارند ، معماری 1 ردیف به دلیل کمتر فرآیند درگیر عملکرد بهتری دارد. در مقایسه با یک فرآیند ، مرزهای فرایند عبور ارتباطات پیچیده تر و کندتر است ، صرف نظر از اینکه از چه نوع تکنیکی IPC (ارتباط بین فرآیندی) استفاده شده است: TCP / IP ، به نام لوله ، صف پیام یا حافظه مشترک و غیره از این رو در یک مستقر برای به دست آوردن عملکرد ، باید تعداد پردازش های برنامه را تا حد ممکن تعداد موارد را حفظ کنیم. چگونه به این هدف برسیم؟ معماری N-Tier به گونه ای قابل اجرا است که جابجایی بین معماری های مختلف لایه به آسانی فقط به روز کردن فایل پیکربندی است. این با جزئیات در مقاله برنامه کاربردی ما معرفی شده است: نمونه معماری N-Tier با ASP.NET MVC3 ، WCF و Entity Framework.

Practically, there are many variations of N-Tier architecture; they are existing for reasons. For example, one possibility is to put the client presenter layer and business layer in the same process to achieve a better interactive performance. You can explore and research this topic further by yourself if you are interested in it.

از نظر عملی ، تغییرات بسیاری در معماری N-Tier وجود دارد. آنها به دلایل موجود هستند برای مثال ، یک امکان این است که لایه ارائه دهنده مشتری و لایه تجاری را در همان فرآیند قرار دهید تا به عملکرد تعاملی بهتری دست یابد. در صورت علاقه به این موضوع می توانید توسط خودتان بیشتر تحقیق و تحقیق کنید.

In addition, 3-Tier architecture can be deployed as 3 or fewer tiers. But a 2-Tier architecture cannot be deployed as 3-Tier; otherwise it should be called 3-Tier architecture, not 2-Tier.

علاوه بر این ، معماری 3 لایه را می توان به عنوان 3 یا چند لایه مستقر کرد. اما یک معماری 2 لایه را نمی توان به صورت 3 طبقه مستقر کرد. در غیر این صورت باید آن را معماری 3 لایه نامید ، نه 2-ردیف.